Inf1-FP Revision Tutorial 6

Comprehensions, recursion and higher-order

Week of 16 – 20 Nov 2015

All of the following questions are taken directly from past exam papers. To check your solutions write tests that they behave according to the examples given, plus QuickCheck tests to test correctness.

1. (a) Write a polymorphic function `f1 :: [a] -> [a]` that returns every third element in a list, starting with the first. For example:

   ```
   f1 "abcdefghij"   ==  "adgj"
   f1 [1,2,3,4,5]    ==  [1,4]
   f1 [0,0,0,0,0]    ==  [0,0]
   f1 []             ==  []
   ```

   Your function may use *basic functions*, *list comprehension*, and *library functions*, but not recursion.

   (b) Write a second function `g1 :: [a] -> [a]` that behaves like `f1`, this time using *basic functions* and *recursion*, but not list comprehension or library functions.

   (c) Write a third function `h1 :: [a] -> [a]` that also behaves like `f1`, this time using one or more of the following higher-order library functions:

   ```
   map    :: (a -> b) -> [a] -> [b]
   filter :: (a -> Bool) -> [a] -> [a]
   foldr  :: (a -> b -> b) -> b -> [a] -> b
   ```

   You may also use *basic functions*, but not list comprehension, other library functions, or recursion.

2. (a) Write a polymorphic function `f2 :: [a] -> [a]` that swaps every two items in a list. Your function should swap the first with the second item, the third with the fourth, and so on. You may assume that the length of an input list is even. For example:

   ```
   f2 "swapping" == "wspaipgn"
   f2 [1,2,3,4]  == [2,1,4,3]
   f2 []         == []
   ```

   Your function may use *basic functions*, *list comprehension*, and *library functions*, but not recursion.

(b) Write a second function `g2 :: [a] -> [a]` that behaves like `f2`, this time using *basic functions* and *recursion*, but not list comprehension or other library functions.

(c) Write a third function `h2 :: [a] -> [a]` that also behaves like `f2`, this time using one or more of the following higher-order library functions:

```
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
foldr :: (a -> b -> b) -> b -> [a] -> b
```

You may also use *basic functions*, but not list comprehension, other library functions, or recursion.

3. (a) Write a function `f3 :: [Int] -> Int` that computes the sum of the products of adjacent elements in a list of even length, as shown below. The function should give an error if provided a list of odd length. For example:

```
    f3 [1,2,3,4]        =  1*2 + 3*4            =  14
    f3 [3,5,7,5,-2,4]   =  3*5 + 7*5 + (-2)*4   =  42
    f3 []                                       =  0
    f3 [1,2,3]                                  =  error
```

Use *basic functions*, *list comprehension*, and *library functions*, but not recursion.

(b) Write a second function `g3 :: [Int] -> Int` that behaves like `f3`, this time using *basic functions* and *recursion*, but not list comprehension or other library functions.

(c) Write a third function `h3 :: [Int] -> Int` that also behaves like `f3`, this time using one or more of the following higher-order library functions:

```
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
foldr :: (a -> b -> b) -> b -> [a] -> b
```

You may also use *basic functions*, but not list comprehension, other library functions, or recursion.